



Patrón Singleton

Juan David Sánchez Aroca
Juan Camilo Correa Pacheco
Ing. Software II
Universidad del Quindío

¿Que es el patrón singleton ?

- Es un patrón diseñado para limitar la creación de objetos pertenecientes a una clase. El objetivo de este patrón es el de garantizar que una sola clase tenga una instancia (o ejemplar) y proporcionar un punto de acceso global a ella. Este patrón; por ejemplo, suele ser utilizado para las conexiones a bases de datos.

Implementar patrón singleton

- Este patrón se implementa haciendo privado el constructor de la clase y creando un atributo estático que contiene la instancia de la clase el cual puede ser accedido desde su respectivo getter y setter .
- Tipos de Singleton: Singleton con información de inicialización y Singleton con creación diferida

Información global de la aplicación

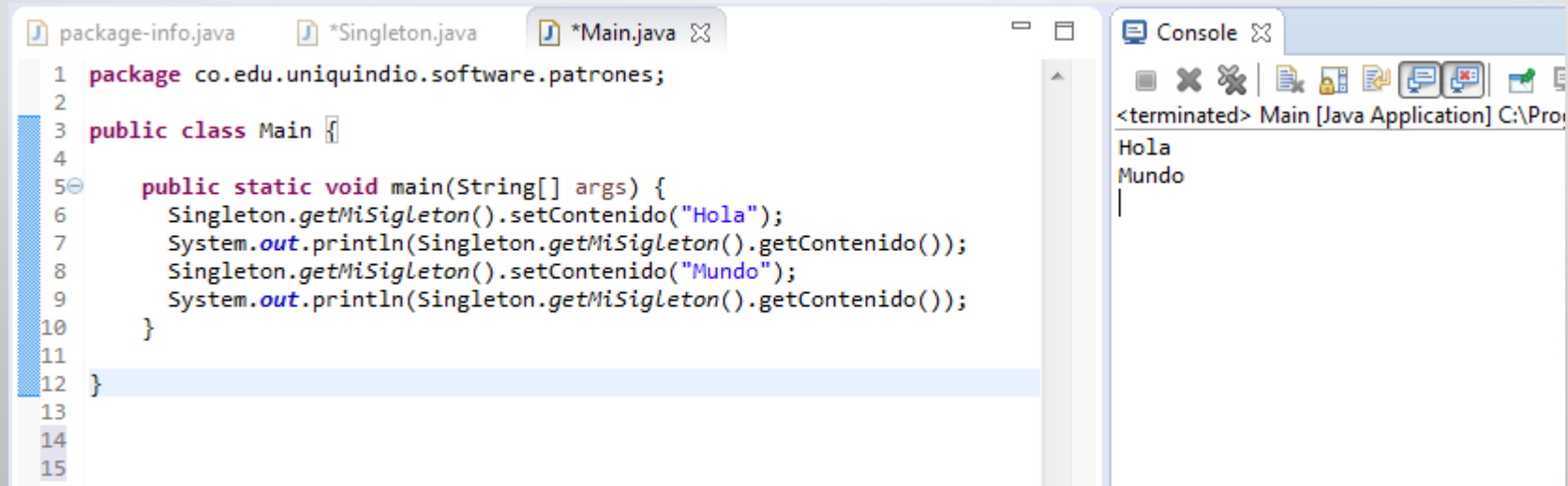
- En muchas ocasiones queremos almacenar información global de una aplicación, accesible desde cualquier punto.
- Podríamos agruparla en un objeto de una clase, y pasar su referencia al resto de objetos.
- En ocasiones resulta complejo pasar esta referencia.

```
package co.edu.uniquindio.software.patrones;

public class Singleton {

    private String contenido;

    private static Singleton miSingleton=new Singleton();
    private Singleton(){
        super();
    }
    public String getContenido() {
        return contenido;
    }
    public void setContenido(String contenido) {
        this.contenido = contenido;
    }
    public static Singleton getMiSingleton() {
        return miSingleton;
    }
    public static void setMiSingleton(Singleton miSingleton) {
        Singleton.miSingleton = miSingleton;
    }
}
```



The screenshot shows an IDE with three tabs: package-info.java, *Singleton.java, and *Main.java. The *Main.java tab is active, showing the following code:

```
1 package co.edu.uniquindio.software.patrones;
2
3 public class Main {
4
5     public static void main(String[] args) {
6         Singleton.getMiSingleton().setContenido("Hola");
7         System.out.println(Singleton.getMiSingleton().getContenido());
8         Singleton.getMiSingleton().setContenido("Mundo");
9         System.out.println(Singleton.getMiSingleton().getContenido());
10    }
11
12 }
13
14
15
```

To the right of the code editor is a console window titled "Console". It shows the output of the program:

```
<terminated> Main [Java Application] C:\Pro
Hola
Mundo
|
```

```

public class Saldo {

    private static Saldo INSTANCIA = new Saldo();
    // En Android casi siempre necesitas el contexto
    private Context contexto;
    // Otras variables de la clase
    private int saldo = -1;

    private Saldo() {}

    public static Saldo getInstancia() {
        return INSTANCIA;
    }

    ...

```

```

...
// Método para inicializar el objeto
public void inicializa(Context contexto){
    this.contexto = contexto;
    SharedPreferences pref = contexto.getSharedPreferences(
        "pref", Context.MODE_PRIVATE);
    int saldo = pref.getInt("saldo_inicial", -1);
}

public int getSaldo() {
    return saldo;
}

public void putSaldo(int saldo) {
    this.saldo = saldo;
}
}

```



```

Saldo saldo = Saldo.getInstancia();
saldo.inicializa(contexto);
int n = saldo.getSaldo();

```

```
public class Singleton {  
    private static Singleton INSTANCIA = null;  
    private Singleton() {...}  
  
    public static Singleton getInstancia() {  
        if (INSTANCIA == null) {  
            synchronized (Singleton.class) {  
                if (INSTANCIA == null) {  
                    INSTANCIA = new Singleton();  
                }  
            }  
        }  
        return INSTANCIA;  
    }  
}
```